Constructor and Type of Constructors

A constructor is a kind method whose name is same as a class name does not have return type.
It is used for initializing an object's non-static variables with given values.
In Java, constructor role is only initializing object, object is created by new operator.

Rules in Defining Constructor

1. Constructor name should be same as class name.
2. It should not contain return type.
3. It should not contain executable modifiers
4. In its logic return statement with value is not allowed

Note:

1. It can have all four accessibility modifiers.
2. It can have parameters.
3. It can have throws clause - it means we can throw exception from constructor.
4. It can have logic, as part of logic it can have all Java legal statement except return statement with value.
5. We can place return; in constructor.

Constructor creation **syntax**

```
class Example {

    Example() {

        ---------
        ---------
        ---------

    }
}
```

All Java legal statements are allowed except return statement with value.

Below application shows defining a class with constructor and its execution

```
class Example {
    Example() {
        System.out.println("constructor");
    }
    public static void main(String[] args) {

        System.out.println("main");
        Example e = new Example();
    }
}
```

O/P:

```
main
constructor
```

Rules in calling constructor

Constructor must be called along with "*new*" keyword; else it leads to compile time error.

Check below program

```
class Example {
        Example(){
                System.out.println("constructor");
        }
        public static void main(String[] args) {
                System.out.println("main");

                Example e = new Example();

                Example();
        }
}
```

CE: cannot find symbol
symbol: method Example()

Q) Can we define a method with same class name?

Yes, it is allowed to define a method with same class name. No compile time error and no runtime error is raised, but it not recommended as per coding standards.

Q) If we place return type in constructor prototype will leads to CE?

No, because compiler and JVM consider it as method.

Below program shows defining a method with the same class name

```
public class Example
{
    Example()
    {
        System.out.println("In Constructor");
    }
    void Example()
    {
        System.out.println("In method");
    }
    public static void main(String[] args)
    {
        Example e = new Example();   <- Constructor call
        e.Example();                 <- method call
    }
}
```

O/P

```
In constructor
In method
```

Q) How compiler and JVM can differentiate constructor and method definitions if both have same class name?

By using *return type*, if there is a return type it is considered as method else it is considered as constructor.

Q) How compiler and JVM can differentiate constructor and method invocations if both have same class name?

By using *new* keyword, if new keyword is used in calling then constructor is executed else method is executed.

Q) Why return type is not allowed for constructor?

As there is a possibility to define method with same class name, return type is not allowed to constructor to differentiate constructor block from method block.

Q) Why constructor name should be same as class name?

Every class object is created by using the same new keyword, so it must have information about the class to which it must create object. For this reason constructor name should be same as class name.

Q) Can we declare constructor as private?

Yes, we can declare constructor as private. All four accessibility modifiers are allowed to constructor. We should declare constructor as private for not to allow user to create object from outside of our class. Basically we will declare constructor as private to implement singleton design pattern.

Q) Is constructor definition mandatory in class?

No, it is optional. If we do not define constructor compiler define constructor.

Check below diagram

Example.java Developer written code

```
class Example
{
        public static void main(String[] args)
        {
                System.out.println("main");
        }
}
```

As you can observe in the above diagram, constructor is automatically placed by compiler in Example.class.

Example.class Compiler changed code

```
class Example
{
    Example()
    {
        super();
    }
        public static void main(String[] args)
        {
                System.out.println("main");
        }
}
```

It is placed by ***compiler*** automatically

Q) Can we define empty class? If so, is it really an empty class?

Yes we can define empty class, but after compilation it has constructor definition that is placed by compiler. So really it is not an empty class.

Check below diagram

Example.java Developer written code

```
class Example {

}
```

Example.class Compiler changed code

```
class Example extends java.lang.Object {
    Example(){
        super();
    }
}
```

It is placed by ***compiler*** automatically

Q) How can we know that compiler places constructor in class file?

We should use "**javap**" tool. It is a java binary file available in "jdk\bin" folder. It is used to decompile class file to check that class members.

After compilation use ***javap*** tool as shown below

```
C:\Windows\system32\cmd.exe
D:\NareshIT\JavaHari\NonStaticMembers:javac Example.java

D:\NareshIT\JavaHari\NonStaticMembers:javap Example
Compiled from "Example.java"
class Example extends java.lang.Object{
    Example();
}

D:\NareshIT\JavaHari\NonStaticMembers:_
```

Note: javap tool only displays class member's prototype. It does not show body and logic.

Types of constructors

Java supports 3 types of constructors

1. Default constructor
2. No-argument | non-parameterized constructor
3. Parameterized constructor

- The compiler given constructor is called default constructor. It does not parameters and logic except *super()* call.

- The developer given constructor without parameters is called no-arg | non-parameterized constructor.

- The developer given constructor with parameters is called parameterized constructor. In developer given constructor we can have logic.

Rule: *super()* call must also be placed as a first statement in developer given constructors. If developer does not place it, compiler places this statement.

Q) Why compiler given constructor is called default constructor?

Because

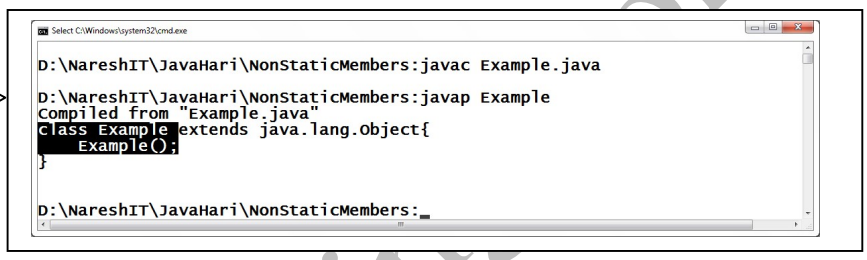
1. It is by default added by compiler
2. Its name is same as its class name
3. It does not have parameters and logic
4. Its accessibility modifiers are same as class accessibility modifiers

Q) What is the default accessibility modifier of a default constructor?

It is assigned from its class. So, it may be default or public. Check below programs

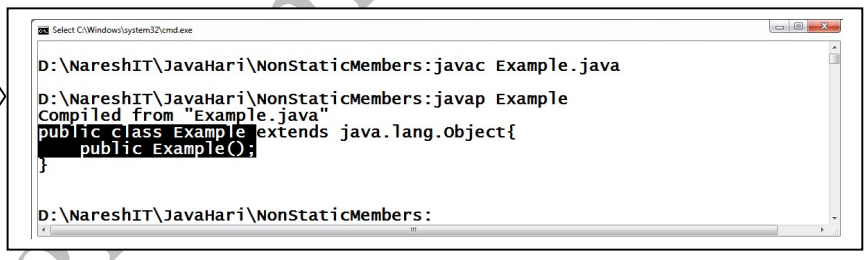
Case #1: class is created with default accessibility modifier, so constructor is also created with default accessibility modifier

```
class Example {

}
```



Case #2: class is created with public accessibility modifier, so constructor is also created with public accessibility modifier

```
public class Example {

}
```



Q) Why compiler defines default constructor without logic and parameters?

Because compiler does not know logic required for your class. The object's initialization logic is different from one class to another class.

But it places super() method call in all constructors because it is generic logic required for every class for calling super class constructor in order to initialize super class non-static variables when subclass object is created.

Since there is no logic in default constructor compiler defines constructor without parameters.

Q) When does compiler provide default constructor?,

Only if there is no explicit constructor defined by developer.

Q) When developer must provide constructor explicitly?

If we what to execute some logic at the time of object creation, that logic may be object initialization logic or some other useful logic.

Find out if there are any compile time errors in the below cases. If there are no compile time errors identify what is the constructor used in object creation.

Before answering below cases, first compile it yourself and add missing code to your program that is placed by complier, after that try executing the program yourself to get output. Habituate this procedure always to get right output.

Case #1:

```
class Example{
        public static void main(String[] args) {
                System.out.println("In main");
                Example e = new Example();
        }
}
```

Case #2:

```
class Example{
        Example(){
                System.out.println("In no-arg constructor");
        }
        public static void main(String[] args) {
                System.out.println("In main");
                Example e = new Example();
        }
}
```

Case #3:

```
class Example{
        Example(){
                System.out.println("In no-arg constructor");
        }
        Example(int x){
                System.out.println("In int-arg constructor");
        }

        public static void main(String[] args) {
                System.out.println("In main");
                Example e = new Example();
        }
}
```

Case #4:

```
class Example{
        Example(int a){
                System.out.println("In int-arg constructor");
        }
        public static void main(String[] args)  {
                System.out.println("In main");
                Example e = new Example();
        }
}
```

Case #5:

```
class Example{
        Example(int a){
                System.out.println("In int-arg constructor");
        }
        public static void main(String[] args)  {
                System.out.println("In main");
                Example e = new Example(50);
        }
}
```

Answers

Case #1:

NO CE, NO RE,
output:
In main
Object is created using default constructor.

Case #2:

NO CE, NO RE,
output:
In main
In no-arg constructor
Object is created using no-arg constructor.

Case #3:

NO CE, NO RE,
output:
In main
In no-arg constructor
Object is created using no-arg constructor.

Case #4:

This case compilation leads to CE:
cannot find symbol
Symbol: constructor Example()

In this class we do not have no-arg constructor and also default constructor. Compiler will not provide default constructor because this class has excplit constructor.

Solution: object should be created only by using parameterized constructor as shown below in

Case 5

Parameterized constructor is called by passing its parameter type argument. So, NO CE, NO RE

Q) If class has explicit constructor, does it has default constructor?

No. Compiler places default constructor only if there is no explicit developer given constructor.

Conclusion: Q) **How can we create object of a class?**
We must create object of a class by using new keyword and available constructor.

Q) Can we consider both default and no-arg constructors are same?
No, both are different. They are seems to be same, but really not same.

Q) What are the differences between no-argument and default constructor?

Default constructor	No-arg constructor
1. It is given by compiler	1. It is given by developer
2. It accessibility modifier is same as class accessibility modifier. So the only allowed accessibility modifiers are *default* or *public*.	2. It can have all four accessibility modifiers as it is defined by developer. So the allowed accessibility modifiers are *private, default, protected, public*
3. It does not have logic except super() call.	3. It can have logic including super() call.

Note: The common point between these two constructors is "both constructors do not have parameters.

Q) Can we define a class with public accessibility modifier and a constructor with other accessibility modifier?
A) Yes we can define. Compiler does not change developer given constructor accessibility modifier to class accessibility modifier.

Q) When should we define parameterized constructor in a class?
To initialize object dynamically with user given values then we should defined parameterized constructor.

What is the difference we can find in objects creation by using all three constructors?
Case 1: object creation with default constructor
The objects created by using default constructor will have same default assigned values.

```
class Example {
        int x = 10, y = 20;

        public static void main(String[] args) {
                Example e1 = new Example();
                Example e2 = new Example();
        }
}
```

In the above case both objects have same default assigned values 10, 20.

Case 2: object creation with no-arg constructor
The objects created by using no-arg constructor will have same values initialized in constructor.

```
class Example {
	int x = 10, y = 20;
	Example(){
		x = 50;
		y = 60;
	}
	public static void main(String[] args){
		Example e3 = new Example();
		Example e4 = new Example();
	}
}
```

In the above case both objects has same values 50, 60.

Case 3: object creation with parameterized constructor
The objects created by using parameterized constructor will have user given values. Those values may be same or different.

```
class Example{
	int x = 10, y = 20;
	Example(int x, int y){
		this.x = x;
		this.y = y;
	}
	public static void main(String[] args){
		Example e5 = new Example(5, 6);
		Example e6 = new Example(5, 6);
		Example e7 = new Example(7, 8);
	}
}
```

In the above case e5 and e6 objects has same user given values "5, 6" and e7 object will have "7, 8".

Q) How many constructors can be defined in a class?
In a class we can define multiple constructors, but every constructor must have different parameters type (or) parameters order. So in a class we can define one no-argument constructor + 'n' number of parameterized constructors.

Q) What is a constructor overloading?

Defining multiple constructors with different parameter types | order | list is called constructor overloading.

Below program shows implementing constructor overloading

```
class Example
{
        Example(){
                Sopln("no-arg constructor");
        }
        Example(int x){
                Sopln("int-arg constructor");
        }

        Example(String s){
                Sopln("String-arg constructor");
        }
```

```
    public static void main(String[] args){
            System.out.println("main");

            Example e1 = new Example();

            Example e2 = new Example(10);

            Example e3 = new Example("abc");
    }
}
```

O/P

```
main
no-arg constructor
int-arg constructor
String-arg constructor
```